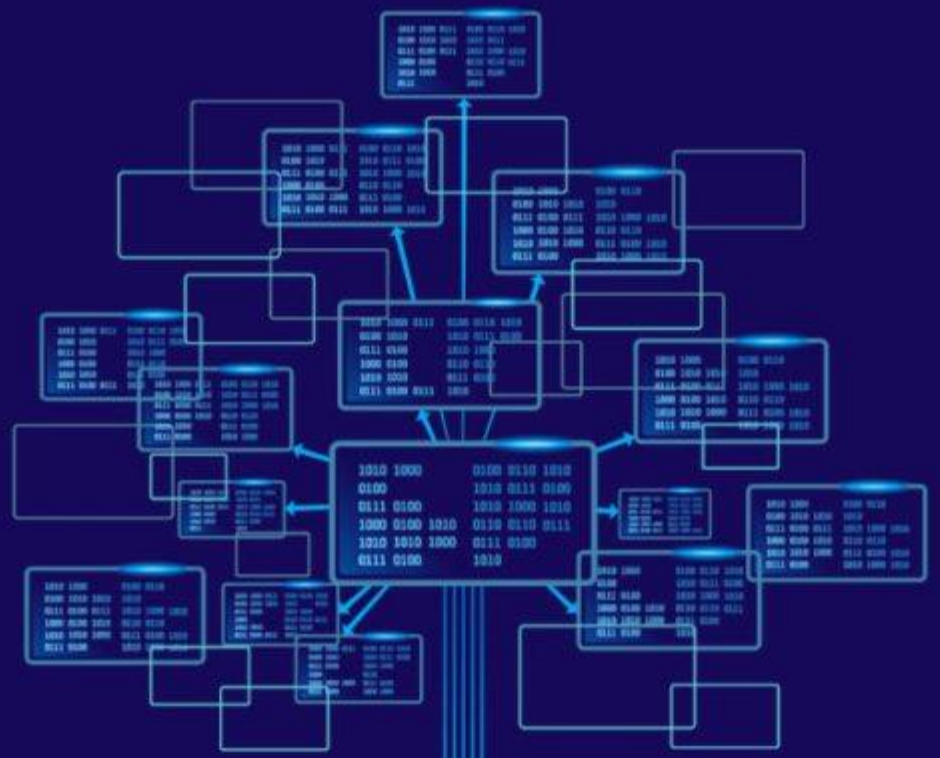


AĞAÇ ALGORİTMALARI

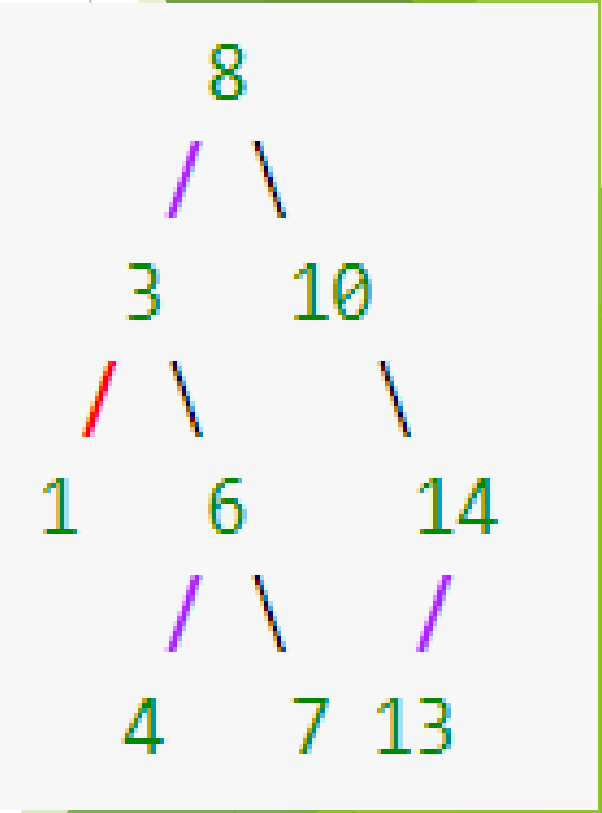
Dr. Öğr. Üyesi Nilgün Şengöz

Ağaç Türleri

- Ağaçlar, bilgisayar biliminde ve matematikte önemli bir veri yapısıdır. Ağaçlar, hiyerarşik ilişkileri modellemek için kullanılır ve birçok algoritma ve veri yapısı için temel oluştururlar. Ağaçlar, bir kök düğüm (root node) ve bu kök düğümden türetilen çeşitli düğümlerden oluşur.
- Ağaçlar, veri depolama, arama, sıralama, yönlendirme (routing) ve yapay zeka gibi birçok alanda yaygın olarak kullanılır. Her bir ağaç türü, belirli bir probleme uygun olan farklı özelliklere ve avantajlara sahiptir.

İkili Arama Ağacı (Binary Search Tree - BST)

- İkili Arama Ağacı (BST), her bir düğümünde bir anahtar değer ve en fazla iki çocuk bulunan bir ağaç veri yapısıdır. BST, sol alt ağaçtaki düğümlerin anahtar değerlerinin, düğümün anahtar değerinden küçük olduğu, sağ alt ağaçtaki düğümlerin ise anahtar değerlerinin düğümün anahtar değerinden büyük olduğu bir şekilde organize edilir. Bu özellik, arama işlemlerini hızlandırır ve BST'yi etkili bir veri yapısı yapar.
- İkili arama ağacında bir düğüm en fazla iki tane çocuğa sahip olabilir ve alt/çocuk bağlantıları belirli bir sırada yapılır.



İkili Arama Ağacı (Binary Search Tree - BST)

- Sonlu düğümler kümesidir. Bu küme boş bir küme olabilir (empty tree). Boş değilse şu kurallara uyar.
- Kök olarak adlandırılan özel bir düğüm vardır
- Her düğüm en fazla iki düğüme bağlıdır
- • Sol Çocuk (Left child) : Bir düğüm'ün (node) sol işaretçisine bağlıdır.
- • Sağ Çocuk (Right child) : Bir düğüm'ün (node) sağ işaretçisine bağlıdır.
- Kök hariç her düğüm bir daldan gelmektedir.
- • Tüm düğümlerden yukarı doğru çıkıldıkça sonuçta köke ulaşılır.

İkili Arama Ağacı (Binary Search Tree - BST)

- Her düğümün sol alt ağacındaki düğümlerin anahtar değerleri, düğümün anahtar değerinden küçüktür.
- Her düğümün sağ alt ağacındaki düğümlerin anahtar değerleri, düğümün anahtar değerinden büyüktür.
- BST, tekrarlanan anahtar değerlerini içermez. Her anahtar değeri yalnızca bir kez bulunur.

İkili Arama Ağacı (Binary Search Tree - BST)

- **Temel Operasyonlar:**

- Ekleme (Insertion): Yeni bir anahtar değeri ağaca eklenirken, BST'nin yapısını koruyacak şekilde yerleştirilir.
- Silme (Deletion): Belirli bir anahtarı ağaçtan çıkarmak, BST'nin yapısını bozmadan gerçekleştirilir.
- Arama (Search): Verilen bir anahtarın ağaçta olup olmadığını kontrol eder.
- En Küçük ve En Büyük Anahtarı Bulma: BST'de en küçük ve en büyük anahtar değerlerini bulma.
- Dolaşma (Traversal): BST'deki tüm düğümleri belirli bir sıra ile ziyaret etme işlemi. (Örneğin, in-order, pre-order, post-order dolaşma)

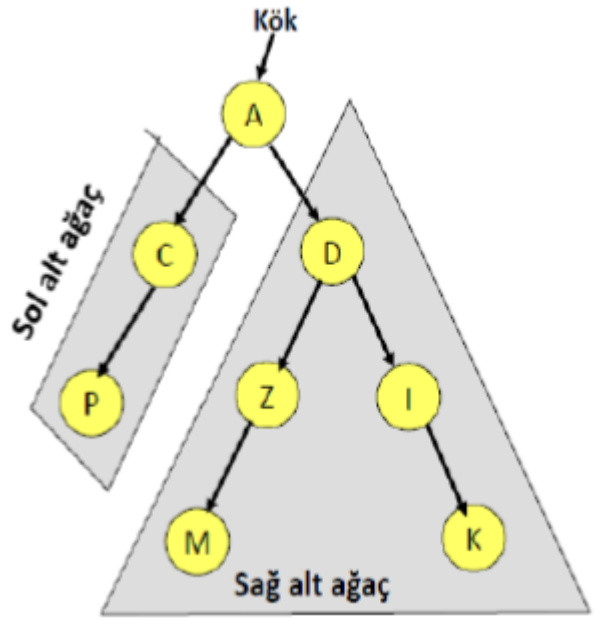
İkili Arama Ağacı (Binary Search Tree - BST)

- BST'nin kullanılabileceği bazı uygulama alanları şunlardır:

- Sıralama: BST, sıralı bir veri kümesini depolamak ve hızlı bir şekilde erişmek için kullanılabilir.
- Veri Tabanları: İndeksleme ve sorgulama işlemleri için kullanılabilir.
- Derin Arama Algoritmaları: Özellikle in-order traversal kullanarak derin arama algoritmaları için temel oluşturabilir.
- Otomatik Tamamlama Sistemleri: Örneğin, bir sözlük veya kod editöründe kod tamamlama.

İkili Arama Ağacı (Binary Search Tree - BST)

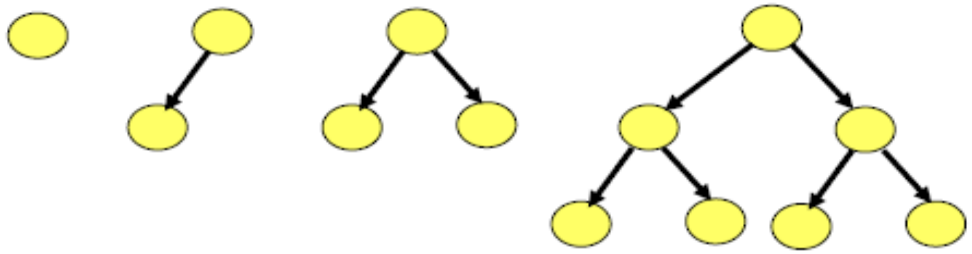
- Alt ağaç kavramı



İki farklı ikili ağaç

İkili Arama Ağacı (Binary Search Tree - BST)

- N tane düğüm veriliyor, İkili ağacın minimum derinliği nedir?

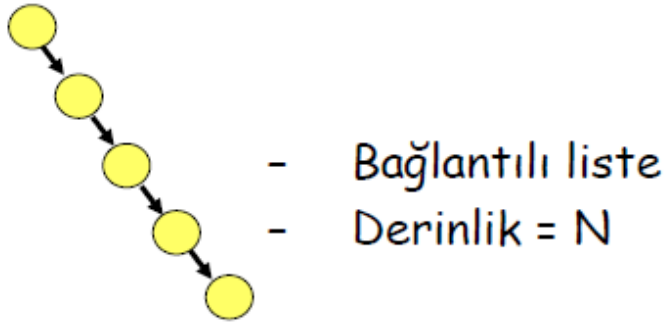


- Derinlik 1: N = 1, 1 düğüm (2^1-1) düğüm
  - Derinlik 2: N = 2, 3 düğüm, (2^2 -1) düğüm
- Herhangi bir d derinliğinde, N = ?
  - Derinlik d: N = 2^d-1 düğüm (tam bir ikili ağaç)
  - En küçük derinlik: Θ(log N)

TANIM: Bir düğümün köke olan uzaklığı derinliktir. Kök düğümün derinliği 1'dir.

İkili Arama Ağacı (Binary Search Tree - BST)

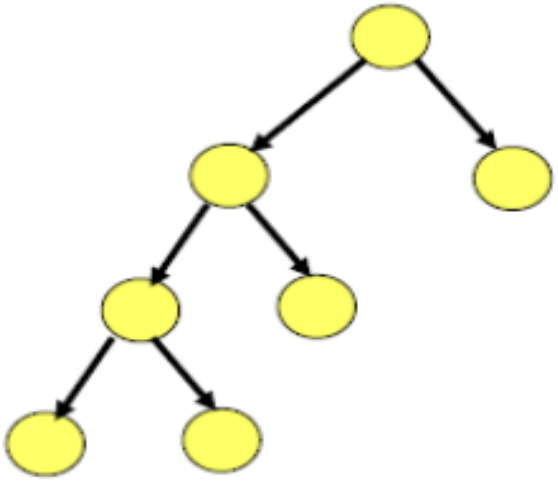
- N düğümlü ikili ağacın minimum derinliği: Θ(log N)
- İkili ağacın maksimum derinliği ne kadardır?
  - Dengesiz ağaç: Ağaç bir bağlantılı liste olursa!
  - Maksimum derinlik = N
  - ✓ **Amaç:** Arama gibi operasyonlarda bağlantılı listeden daha iyi performans sağlamak için derinliğin log N de tutulması gerekmektedir.



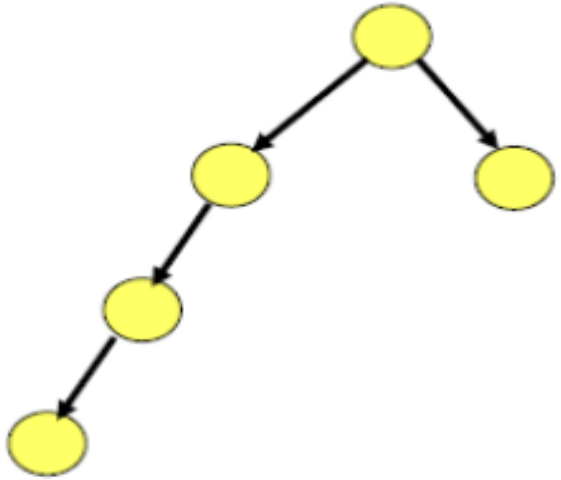
- Bağlantılı liste
- Derinlik = N

Proper (Düzgün) İkili Ağaç (Binary Tree)

- Yaprak olmayan düğümlerin tümünün iki çocuğu olan T ağacı proper(düzgün) binary tree olarak adlandırılır.



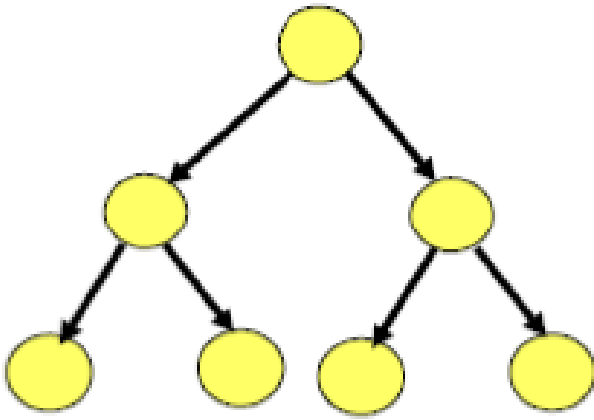
proper(düzgün) binary tree



Improper (düzgün olmayan) binary tree

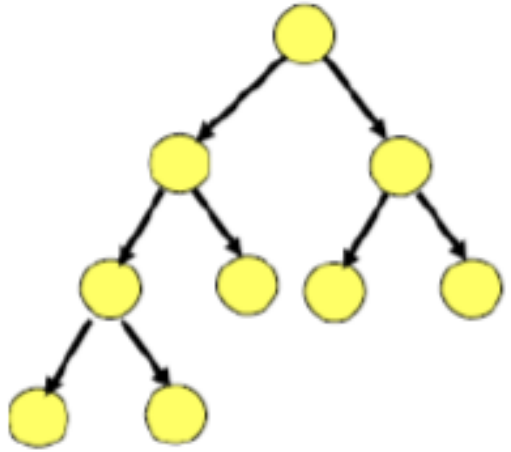
Tam İkili Ağaç (Full Binary Tree)

- Full binary tree:
  - Her yaprağı aynı derinlikte olan
  - Yaprak olmayan düğümlerin tümünün iki çocuğu olan ağaç Full (Strictly) Binary Tree'dir.
- Bir full binary tree'de n tane yaprak varsa bu ağaçta toplam 2n-1 düğüm vardır. Başka bir şekilde ifade edilirse,
  - Eğer T ağacı boş ise, T yüksekliği 0 olan bir full binary ağaçtır.
  - T ağacının yüksekliği h ise ve yüksekliği h'den küçük olan tüm node'lar iki child node'a sahipse, T full binary tree'dir.
  - Full binary tree'de her node aynı yüksekliğe eşit sağ ve sol altağaçlara sahiptir.

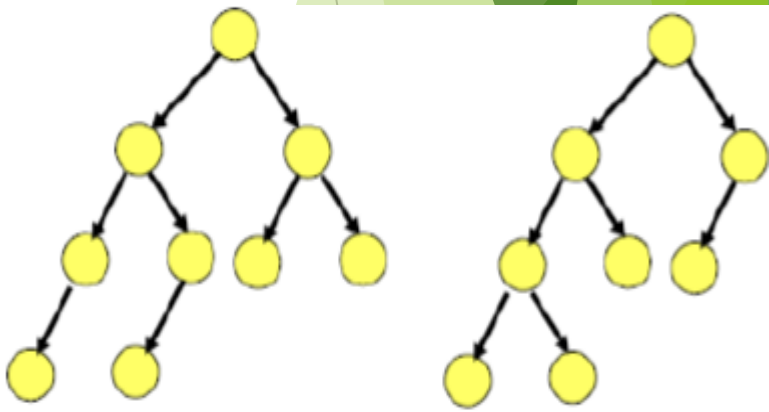


Tamamlanmış İkili Ağaç (Complete Binary Tree)

- Full binary tree'de, yeni bir derinliğe soldan sağa doğru düğümler eklendiğinde oluşan ağaçlara Complete Binary Tree denilir.
  - Böyle bir ağaçta bazı yapraklar diğerlerinden daha derindir. Bu nedenle full binary tree olmayabilirler. En derin düzeyde düğümler olabildiğince soldadır.
  - T, n yükseklikte complete binary tree ise, tüm yaprak node'ları n veya n-1 derinliğindedir ve yeni bir derinliğe soldan sağa doğru ekleme başlanır.
- Her node iki tane child node'a sahip olmayabilir.



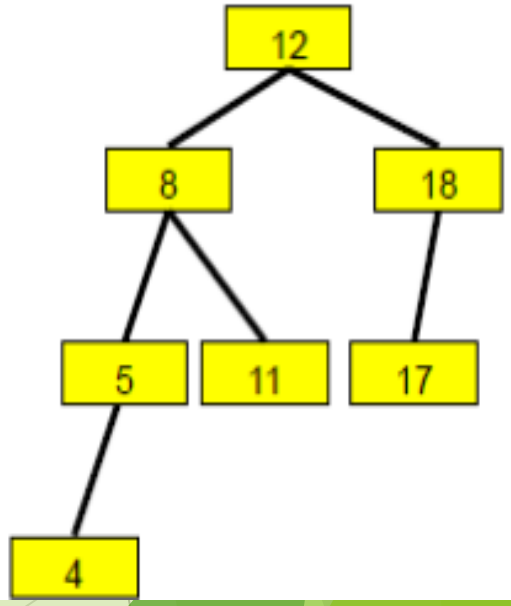
Complete



Incomplete

Dengeli İkili Ağaç (Balanced Binary Tree)

- Yüksekliği ayarlanmış ağaçlardır.
- Bütün düğümler için sol altağacın yüksekliği ile sağ altağacın yüksekliği arasında en fazla bir fark varsa balanced binary tree olarak adlandırılır.
- Complete binary tree'ler aynı zamanda balanced binary tree'dir.
- Her balanced binary tree, complete binary tree olmayabilir. (Neden?)



Dengeli İkili Ağaç (Balanced Binary Tree)

- Her dengeli ikili ağaç (balanced binary tree), tamamlanmış bir ikili ağaç (complete binary tree) olmayabilir çünkü dengeli olmak, her zaman tam bir ağacın tüm özelliklerini sağlamaz.
- Complete bir ikili ağaç, her seviyede tüm düğümlerin dolu olduğu bir ağaçtır. Yani, son seviye hariç her seviyede düğümler doludur ve son seviyede ise düğümler soldan sağa sıralanmıştır.
- Dengeli bir ikili ağaç, her iki alt ağacın yüksekliği arasındaki farkın belirli bir sınıra göre (genellikle 1) sınırlı olduğu bir ağaçtır. Dengeli olmak, ağacın yüksekliğini minimize etmeyi ve bu sayede arama, ekleme ve silme gibi işlemlerin zaman karmaşıklığını optimize etmeyi amaçlar.
- Dolayısıyla, bir ağacın dengeli olması, her seviyede tam olarak dolu düğümlere sahip olmasını garanti etmez. Örneğin, bir dengeli ağacın son seviyesi tam olarak dolu olmayabilir çünkü ağacın sağ ve sol alt ağaçları arasındaki yükseklik farkı nedeniyle bazı düğümler eklenmemiş olabilir. Bu durumda, ağaç tam bir ikili ağaç olmayacaktır.

Heap Tree (Kümeleme Ağacı)

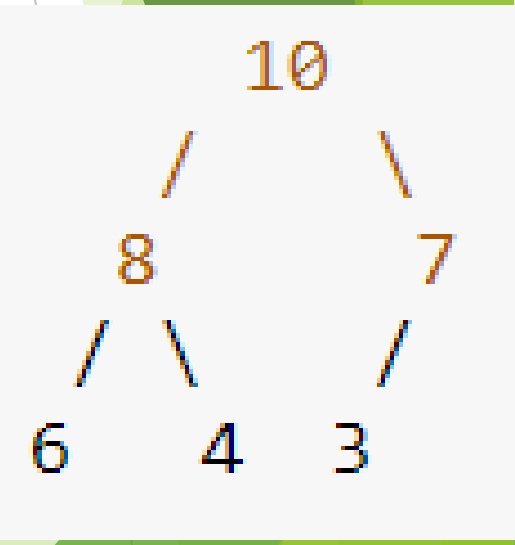
- Kümeleme ağacı (Heap Tree), genellikle bir dizi olarak temsil edilen ve belirli bir özelliği koruyan bir tam ağaç yapısıdır. Bu özellik, genellikle en büyük veya en küçük elemanın (maksimum veya minimum öğe) kök düğümde olmasıdır.
- **Tam Ağaç Yapısı:** Kümeleme ağacı bir tam ağaç yapısına sahiptir. Bu, ağacın tüm seviyelerinin tam olarak dolu olduğu ve son seviyenin sola doğru sıralı düğümlerle doldurulduğu anlamına gelir.
- **Sıralama Özelliği:** Kümeleme ağacında her düğüm, genellikle "maksimum heap" veya "minimum heap" olarak adlandırılan bir sıralama özelliğini korur. Maksimum heap'te, her düğümün değeri alt düğümlerinden büyük veya eşittir. Minimum heap'te ise her düğümün değeri alt düğümlerinden küçük veya eşittir.

Heap Tree (Kümeleme Ağacı)

- Kümeleme ağaçları, sıralama özelliğine göre ikiye ayrılır:
  - Maksimum Heap (Max Heap): Her düğüm, alt düğümlerinden daha büyük veya eşit bir değere sahiptir. Yani, kök düğüm en büyük değere sahiptir.
  - Minimum Heap (Min Heap): Her düğüm, alt düğümlerinden daha küçük veya eşit bir değere sahiptir. Yani, kök düğüm en küçük değere sahiptir.
- Kümeleme ağaçlarında temel işlemler şunlardır:
  - Ekleme (Insertion): Yeni bir eleman kümeleme ağacına eklendiğinde, ağaç yeniden dengelenir ve sıralama özelliği korunur.
  - Silme (Deletion): En büyük veya en küçük eleman kümeleme ağacından çıkarıldığında, ağaç yeniden dengelenir ve sıralama özelliği korunur.
  - En Büyük veya En Küçük Elemanı Alma: Maksimum heap'te en büyük eleman kök düğümde bulunurken, minimum heap'te en küçük eleman kök düğümde bulunur. Bu nedenle, bu işlemler O(1) karmaşıklığında gerçekleştirilebilir.

Heap Tree (Kümeleme Ağacı)

- Kümeleme ağaçları, özellikle öncelik kuyrukları (priority queues) gibi veri yapısı uygulamalarında kullanılır. Öncelik kuyrukları, en yüksek önceliğe sahip öğelerin (maksimum heap kullanılarak) veya en düşük önceliğe sahip öğelerin (minimum heap kullanılarak) hızlı bir şekilde alınması gereken durumlarda kullanılır.
- Ayrıca, kümeleme ağaçları, sıralama algoritmalarında ve grafik algoritmalarında da kullanılabilir.
- Ekleme ve silme işlemleri genellikle O(log n) karmaşıklığına sahiptir, n ağaçtaki eleman sayısını temsil eder.
- En büyük veya en küçük elemanı alma işlemi ise O(1) karmaşıklığına sahiptir.
- Bu ağaçta, en büyük eleman 10'dur ve tüm düğümler sıralama özelliğini korur.
- Kümeleme ağaçları, sıralı bir yapıya sahip olmaları ve temel işlemleri hızlı bir şekilde gerçekleştirebilmeleri nedeniyle birçok uygulamada kullanılır. Öncelik kuyrukları gibi veri yapısı uygulamaları ve sıralama algoritmaları, kümeleme ağaçlarının tipik kullanım alanları arasındadır.



Sözlük Ağacı (Dictionary Tree)

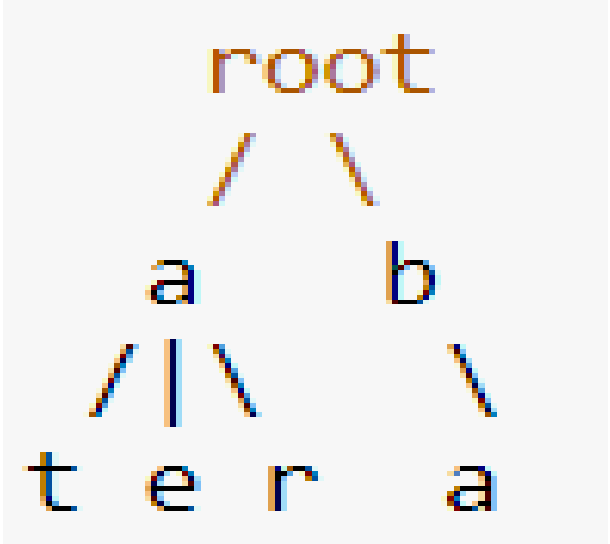
- Sözlük ağacı, metin tabanlı veri yapılarından biridir ve genellikle bir dizi olarak temsil edilen bir veri yapısıdır. Sözlük ağacı, kelime tabanlı arama, ekleme ve silme işlemlerini destekler. Genellikle birçok dil işleme uygulamasında, otomatik tamamlama sistemlerinde ve yazım denetimi gibi alanlarda kullanılır.
- **Temel Özellikler**
- Ağaç Yapısı: Sözlük ağacı, bir ağaç yapısı olarak organize edilmiştir. Her düğüm, bir karakter (harf) veya kelime öbeği içerir.
- Kelime Tabanlı Arama: Sözlük ağacı, kelime tabanlı arama işlemlerini destekler. Kelimeler, ağaçtaki düğümlerde depolanır ve arama işlemi bu düğümlerde gerçekleştirilir.
- Ekleme ve Silme İşlemleri: Yeni kelimeler sözlük ağacına eklenebilir ve var olan kelimeler silinebilir.

Sözlük Ağacı (Dictionary Tree)

- Sözlük ağaçları, genellikle karakter veya kelime tabanlı arama işlemlerini desteklemek için kullanılır. Temel olarak iki tür sözlük ağacı vardır:
- **Trie (Prefix Tree):** Trie, özellikle metin tabanlı arama işlemlerini desteklemek için kullanılan bir tür sözlük ağacıdır. Özellikle kelime önerisi, otomatik tamamlama ve yazım denetimi gibi uygulamalarda kullanılır.
- **Radix Tree (Compact Prefix Tree):** Radix tree, trie yapısının bir türüdür ve daha kompakt bir depolama sağlar. Özellikle büyük veri kümeleri için daha verimli bir seçenektir.
- **Kullanım Alanları**
- Sözlük ağaçları, dil işleme uygulamalarında, otomatik tamamlama sistemlerinde, yazım denetimi ve düzeltme uygulamalarında, anahtar kelime aramalarında ve veri tabanı indeksleme işlemlerinde kullanılır.

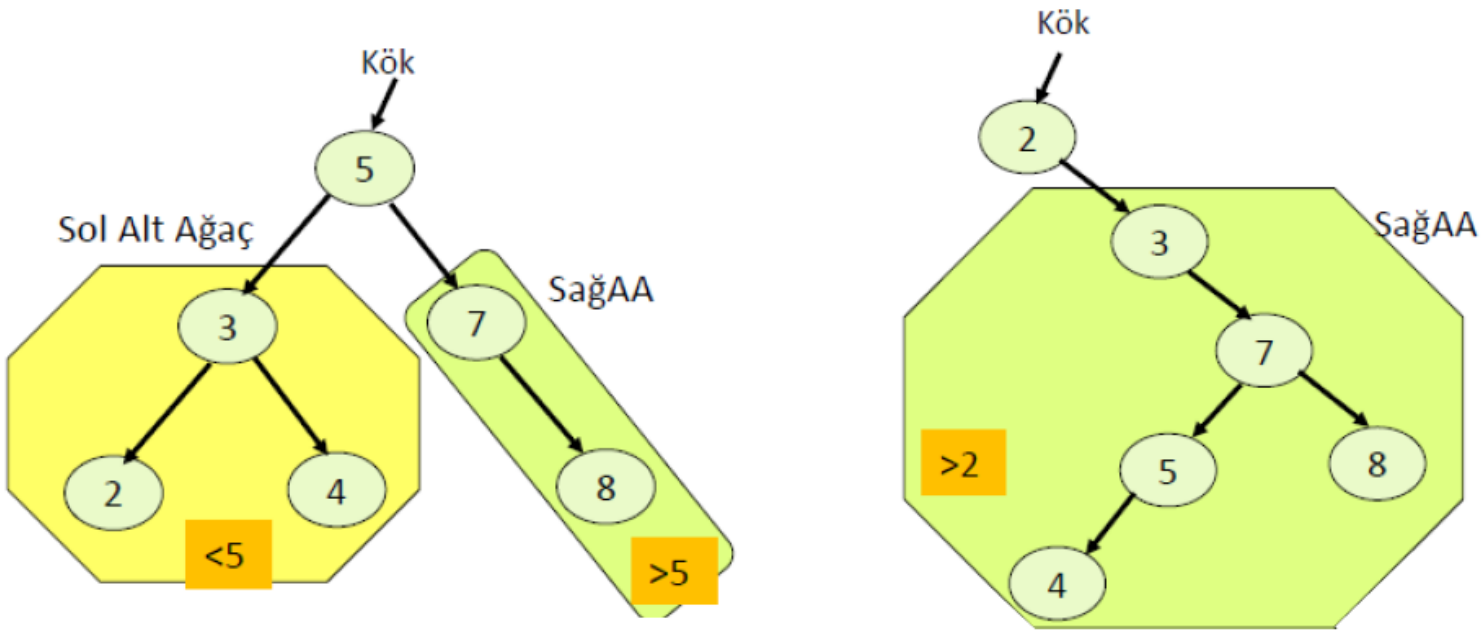
Sözlük Ağacı (Dictionary Tree)

- Bu ağaçta, "ate" ve "bat" gibi kelimeler saklanabilir ve arama işlemleri bu ağaç üzerinde gerçekleştirilebilir.
- Sözlük ağaçları, metin tabanlı veri işleme uygulamalarında yaygın olarak kullanılan veri yapılarından biridir. Kelime tabanlı arama, ekleme ve silme işlemlerini desteklerler ve birçok dil işleme uygulamasında temel bir bileşendir. Trie ve Radix tree gibi türleri, özellikle büyük veri kümeleri için etkili bir şekilde kullanılabilir.



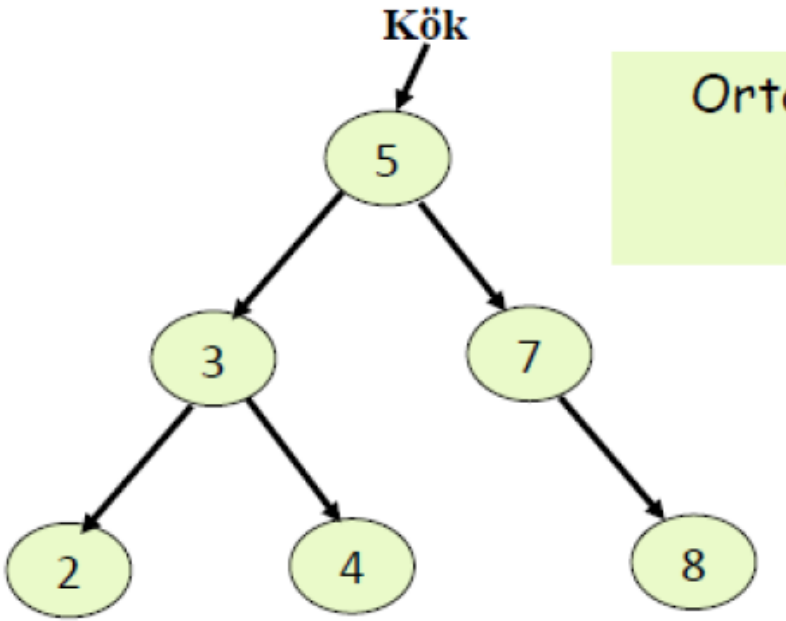
İkili Arama Ağacı (Binary Search Tree - BST)

- İkili Arama Ağacı her bir düğümdeki değerlere göre düzenlenir:
  - Sol alt ağaçtaki tüm değerler kök düğümünden küçüktür.
  - Sağ alt ağaçtaki tüm değerler kök düğümünden büyüktür.



İkili Arama Ağacında Sıralama

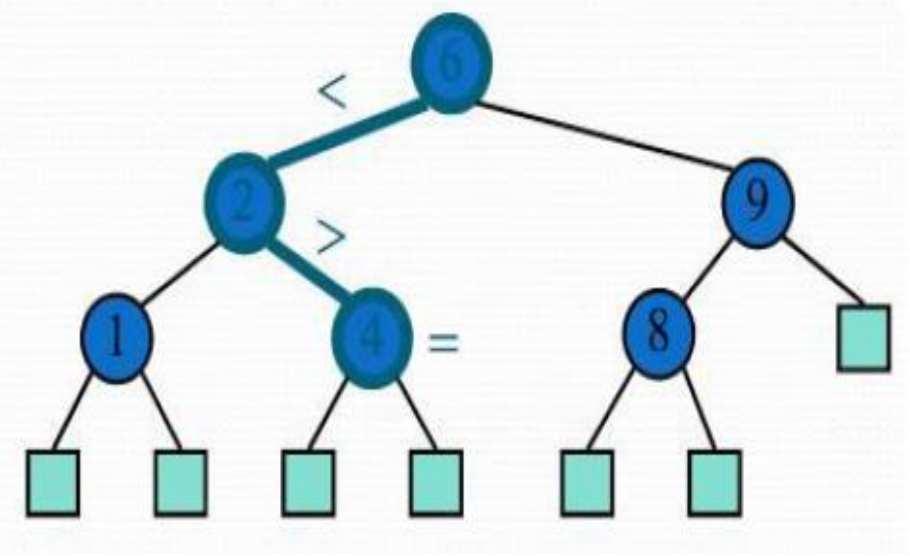
- İkili arama ağacı önemli özelliklerinden birisi Ortada-kök (Inorder) dolaşma algoritması ile düğümlere sıralı bir şekilde ulaşılmasını sağlar.



Ortada-kök sonucu
2 3 4 5 7 8

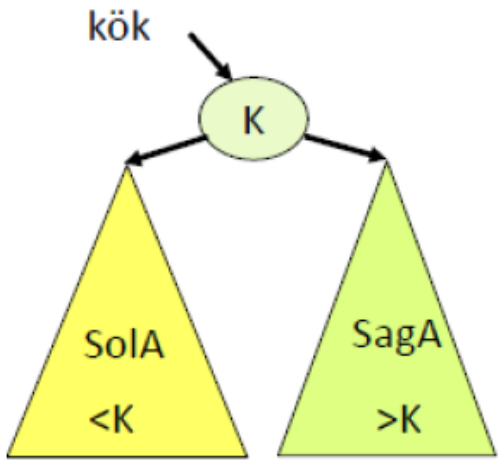
İkili Arama Ağacı (Binary Search Tree - BST) Arama

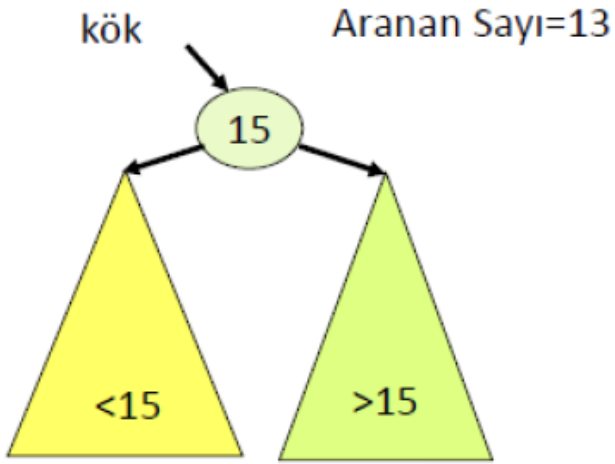
- Bir k anahtarını aramak için, kök düğümden başlayarak aşağı doğru bir yol izlenir.
- Bir sonraki ziyaret edilecek düğüm, k anahtar değerinin geçerli düğümün anahtar değeriyle karşılaştırılması sonucuna bağlıdır.
- Eğer yaprağa ulaşıldıysa anahtar bulunamamıştır ve null değer geri döndürülür.
- Örnek: Bul(4)



İkili Arama Ağacı (Binary Search Tree - BST) Arama

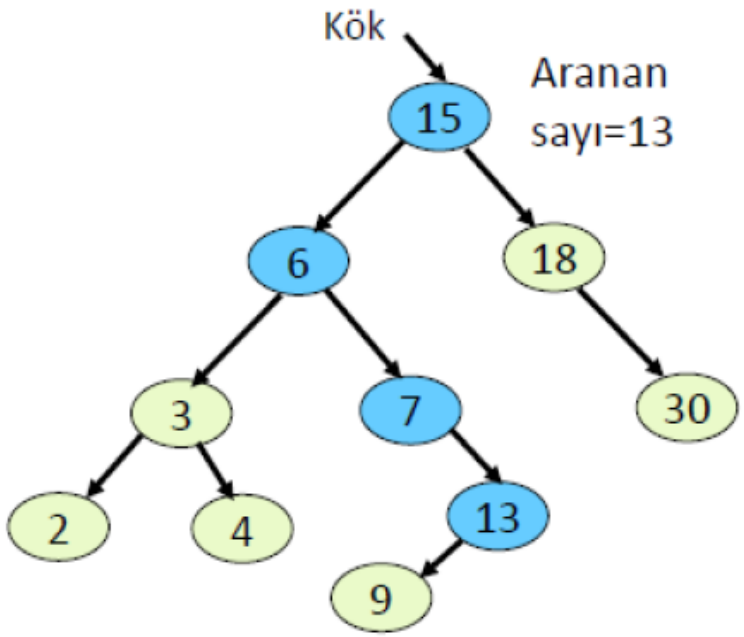
- Değeri içeren düğümü bul ve bu düğümü geri döndür.






```
1.Arama işlemine kökten başla
2.if (aranaDeger == kok.deger) →    return kok;
3.if (aranaDeger < kok.deger)  →    Ara SolAltAğaç
4.else                         →    Ara SagAltAğaç
```

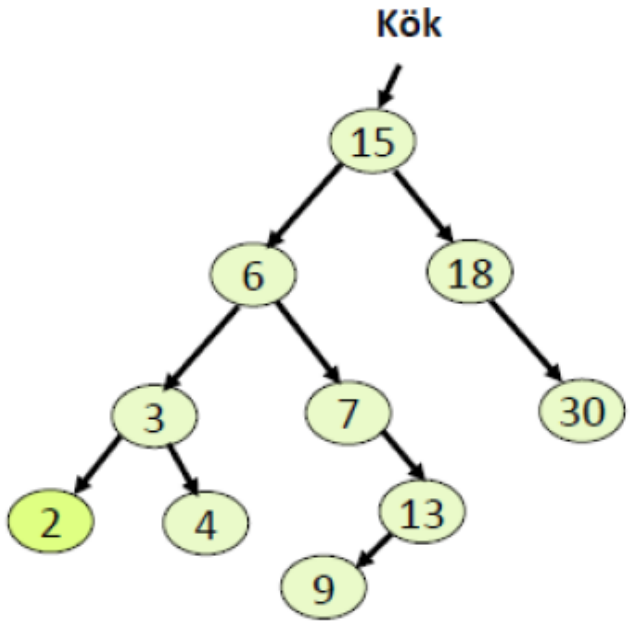
İkili Arama Ağacı (Binary Search Tree - BST) Arama



Mavi renkli düğümler arama sırasında ziyaret edilen düğümlerdir. Algoritmanın çalışma karmaşıklığı O(d)'dir. (d = ağacın derinliği)

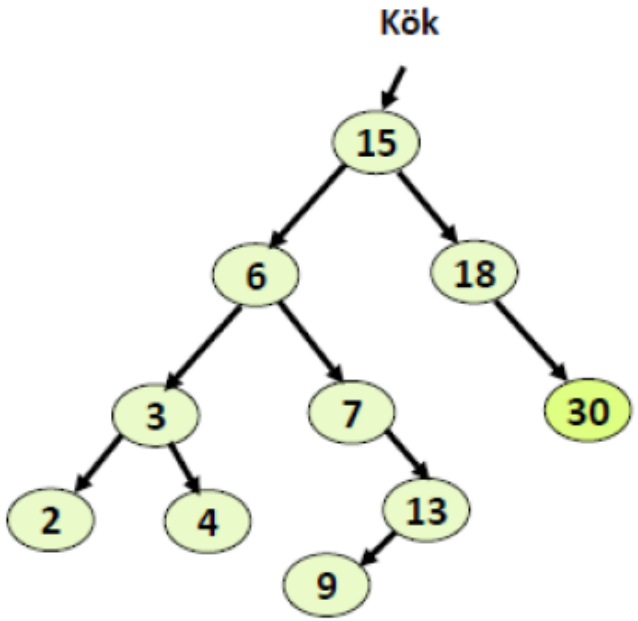
İkili Arama Ağacı (Binary Search Tree - BST) Arama

- Ağaçtaki en küçük elemanı içeren düğümü bulur ve geri döndürür.
  - En küçük elemanı içeren düğüm en soldaki düğümde bulunur.
  - Kökten başlayarak devamlı sola gidilerek bulunur.



İkili Arama Ağacı (Binary Search Tree - BST) Arama

- Ağaçtaki en büyük elemanı içeren düğümü bulur ve geri döndürür.
  - En büyük elemanı içeren düğüm en sağdaki düğümde bulunur.
  - Kökten başlayarak devamlı sağa gidilerek bulunur.



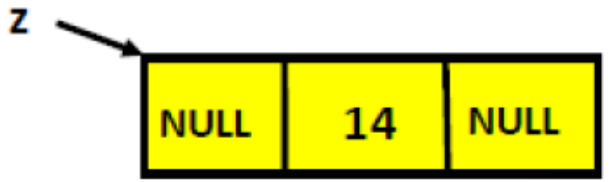
İkili Arama Ağacı (Binary Search Tree - BST) Ekleme

1. Eklenecek değeri içeren “z” isimli yeni bir düğüm oluştur.

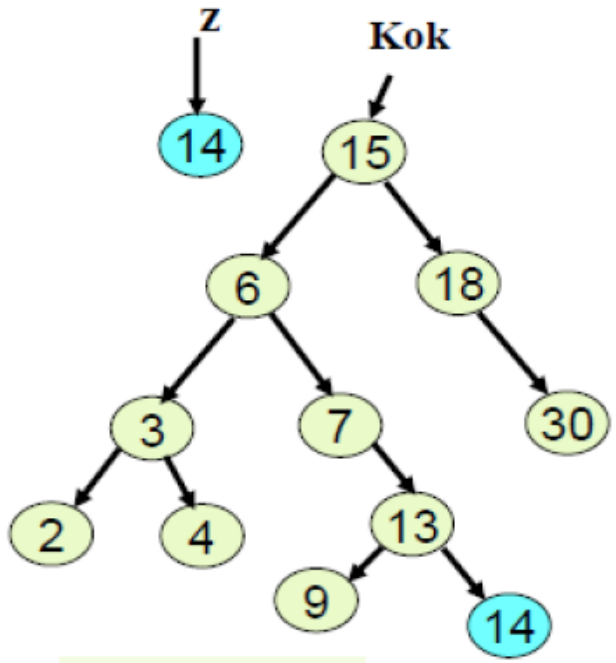
Örneğin: Ekle 14

2. Kökten başlayarak ağaç üzerinde eklenecek sayıyı arıyormuş gibi aşağıya doğru ilerle.

3. Yeni düğüm aramanın bittiği düğümün çocuğu olmalıdır.



Eklenecek «z» düğümü
z.deger=14



Eklemeden sonra

İkili Arama Ağacı (Binary Search Tree - BST) Silme Kaba Kod

while (silinecek düğümün ve ailesinin adresini bulana kadar) {

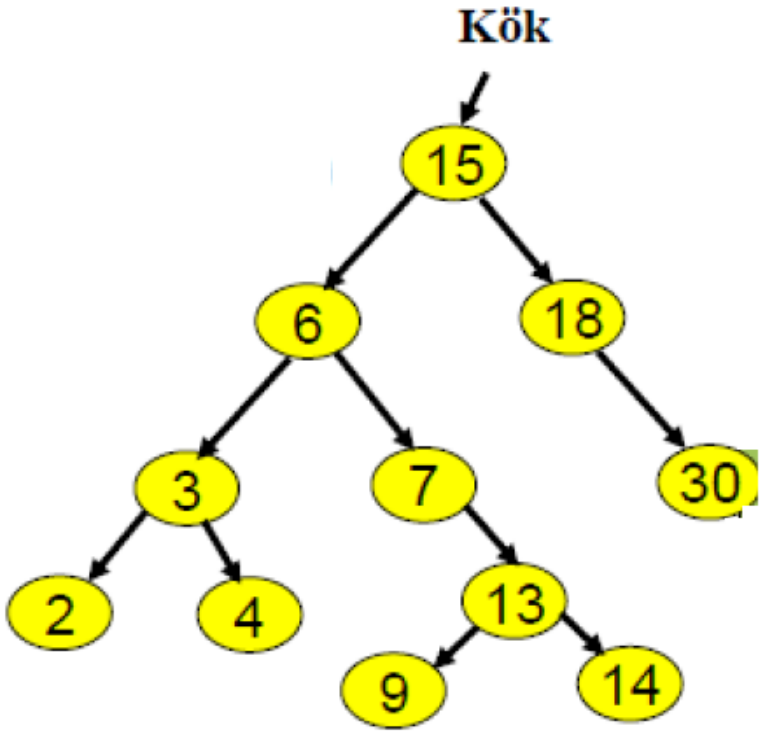
- q <- silinecek düğümün, qa<- ailesinin adresi;
- **if** (silinmek istenen bulunamadı ise)
  - yapacak birşey yok dön;
- **if** (silinecek düğümüm iki alt çocuğu da varsa)
  - sol alt ağacın en büyük değerli düğümünü bul;

  (veya denge bozulmuş ise sağ alt ağacın enküçük değerli düğümünü bul)
  - bu düğümdeki bilgiyi silinmek istenen düğüme aktar;
  - bu aşamada en fazla bir çocuğu olan düğümü sil;
- silinen düğümün işgal ettiği bellek alanını serbest bırak;

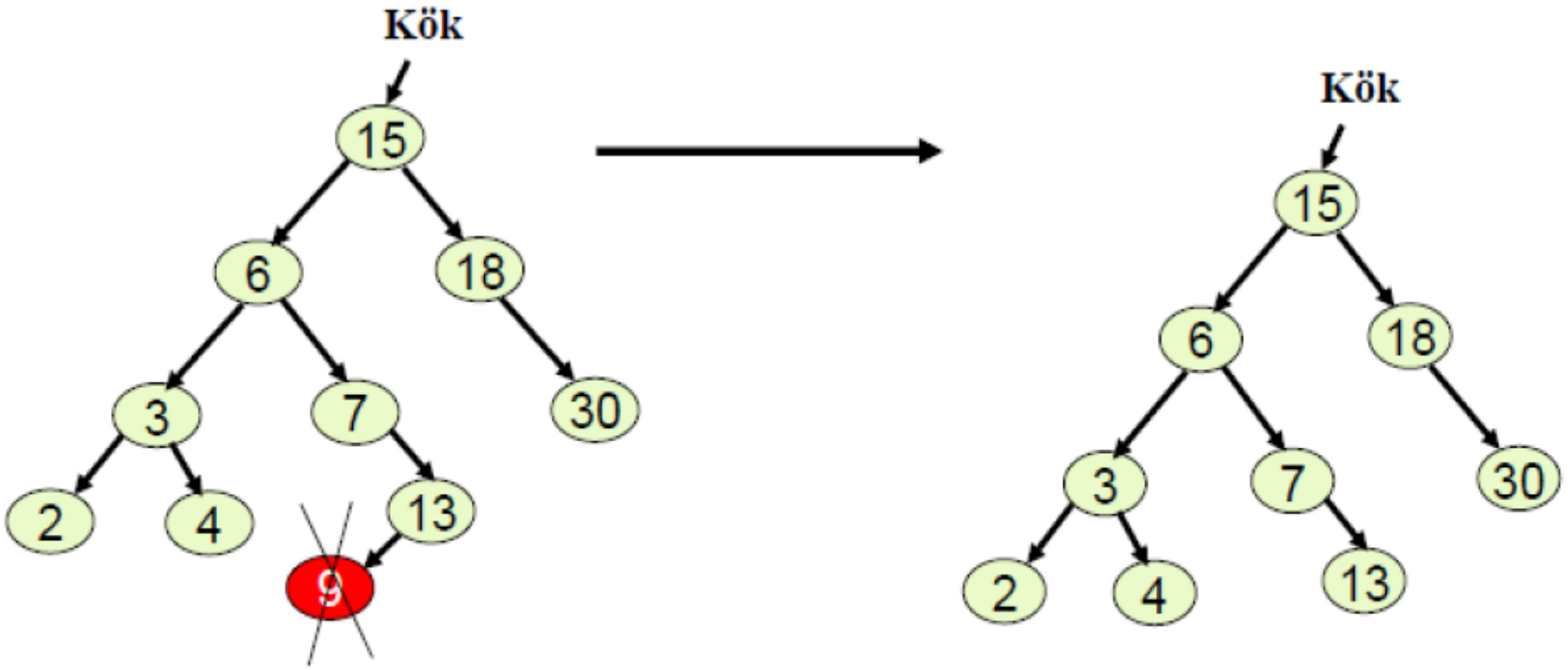
}

İkili Arama Ağacı (Binary Search Tree - BST) Silme

- Silme işlemi biraz karmaşıktır.
- 3 durum var:
  - Silinecek düğümün hiç çocuğu yoksa (yaprak düğüm)
    - Sil 9
  - Silinecek düğümün 1 çocuğu varsa
    - Sil 7
  - Silinecek düğümün 2 çocuğu varsa
    - Sil 6



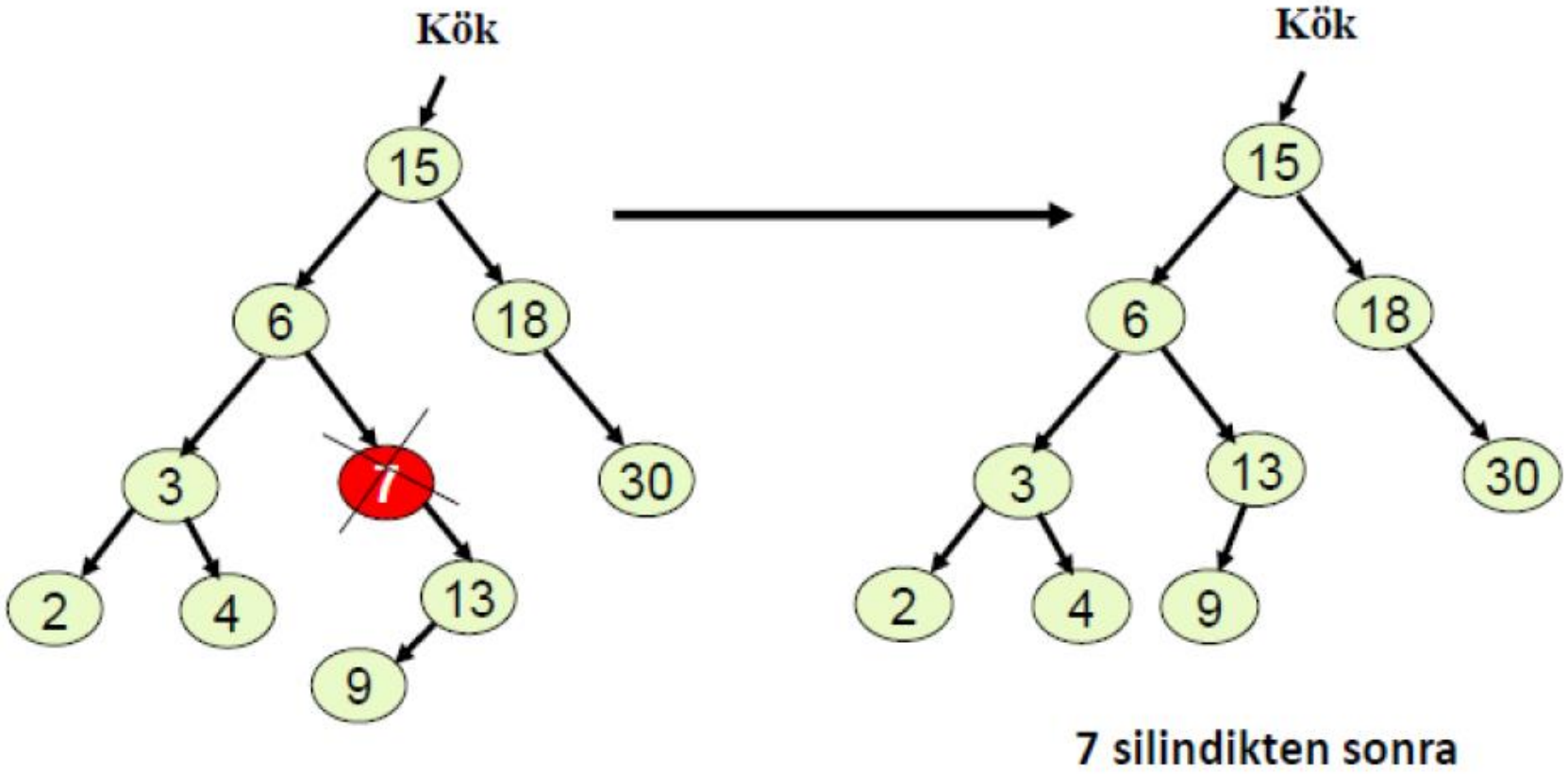
İkili Arama Ağacı (Binary Search Tree - BST) Silme Durum 1- Yaprak Silme



Sil 9: Düğümü kaldırın ve bağlantı kısmını güncelleyin

9 silindikten sonra

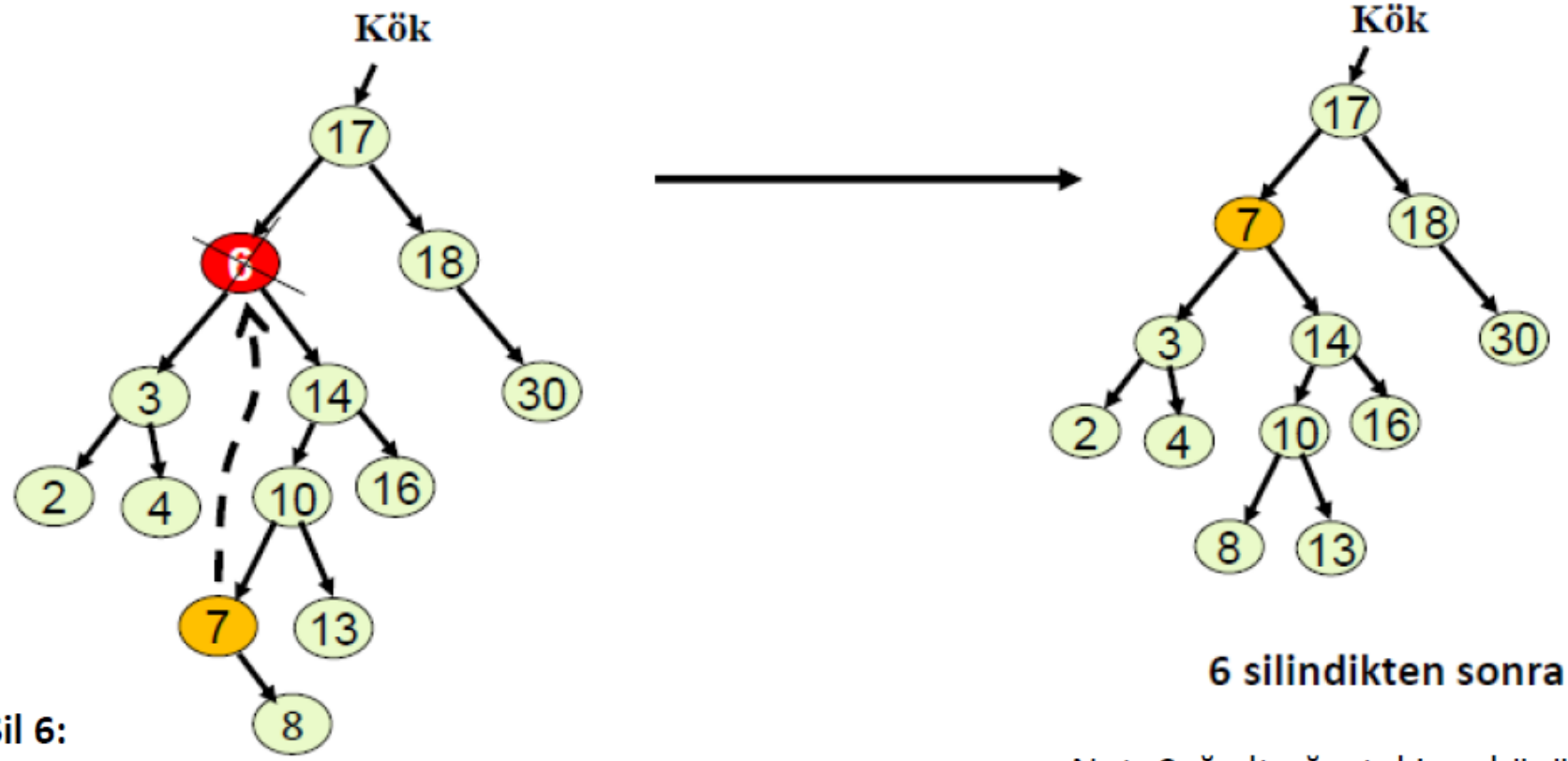
İkili Arama Ağacı (Binary Search Tree - BST)
Silme Durum 2- 1 Çocuklu Düğüm Silme



7 silindikten sonra

Sil 7: Silinecek düğümün ailesi ve çocuğu arasında bağ kurulur

İkili Arama Ağacı (Binary Search Tree - BST)
Silme Durum 3- 2 Çocuklu Düğüm Silme



6 silindikten sonra

Sil 6:

1)Sağ alt ağaçtaki en küçük eleman bulunur.(7)

2)Bu elemanın sol çocuğu olmayacaktır.

3)6 ve 7 içeren düğümlerin içeriklerini değiştirin

4)6 nolu eleman 1 çocuğu varmış gibi silinir.

Not: Sağ alt ağaçtaki en küçük eleman yerine sol alt ağaçtaki en büyük eleman bulunarak aynı işlemler yapılabilir.

İkili Arama Ağacı (Binary Search Tree - BST)

- İkili arama ağacı harita, sözlük gibi birçok uygulamada kullanılır.
  - İkili arama ağacı (anahtar, değer) çifti şeklinde kullanılacak sistemler için uygundur.
    - Örneğin: Şehir Bilgi Sistemi
      - Posta kodu veriliyor , şehir ismi döndürülüyor. (posta kodu/ Şehir ismi)
    - Örneğin: Telefon rehberi
      - İsim veriliyor telefon numarası veya adres döndürülüyor. (isim, Adres/Telefon)
    - Örneğin: Sözlük
      - Kelime veriliyor anlamı döndürülüyor. (kelime, anlam)